
Structure of Classes

CSE 142, Summer 2003
Computer Programming 1

<http://www.cs.washington.edu/education/courses/142/03su/>

Readings and References

- Reading
 - » Chapter 5, *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch
- References
 - » "Language Basics", Java tutorial
<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/index.html>
 - » "javadoc - The Java API Documentation Generator"
<http://java.sun.com/j2se/1.4/docs/tooldocs/windows/javadoc.html>

Basic Class Structure

```
/**  
 * Simple class to illustrate structure of a class definition.  
 */  
public class Acrobat {  
  
    // Properties ...  
  
    //-----  
  
    // Responsibilities ...  
  
}
```

Initial comment describes the overall purpose of the class.

The statements between the curly brackets actually define the class.

Properties are stored as instance variables

```
/**  
 * Simple class to illustrate structure of a class definition.  
 */  
public class Acrobat {  
  
    /** the specific name of this person */  
    private String givenName;  
  
    /** the surname or family name of this person */  
    private String familyName;  
  
    /** the cumulative action count for this Acrobat. */  
    private int actionCount;
```

Responsibilities are implemented in methods

```
/**
 * Twirl around as instructed. Note that since we don't yet
 * have any fancy graphics capabilities our twirling is limited
 * to just saying that we twirled, without any actual twirlosity.
 * @param k the number of times to twirl
 */
public void twirl(int k) {
    System.out.println(familyName+" twirled "+k+" times.");
    actionCount = actionCount + k;
}
```



These are the statements that make up the body of the method.

Structure of a class definition

Instance variables

- Instance variables are the way an object keeps track of its state
 - » One set of instance variables for *each* instance of the class
 - » Each object has its own set of instance variables
- Instance variables are declared outside the body of any constructor or method (but within the definition of the class)
 - » “within the definition of the class” means “between the outside pair of curly braces”
- Instance variables retain their values as long as the object exists

Lifetime of an instance variable

- When an object is created with **new**, Java allocates a chunk of memory for the object
 - » the allocation has space in it for each instance variable
- After allocation the appropriate constructor is called to initialize the instance variables
 - » The programmer is responsible for making sure that the initialization is done correctly and completely
- Instance variables exist from the time an object is created until the time it is destroyed

Classes and objects in memory

Initialization and constructors

- Initialization (another responsibility) is mostly done in constructors

```
/**  
 * Create a new Acrobat using the name information provided.  
 * @param given the specific name of this person  
 * @param family the surname or family name of this person  
 */  
public Acrobat(String given,String family) {  
    givenName = given;  
    familyName = family;  
    actionCount = 0;  
}
```



These are the statements that make up the body of the constructor.

Constructors

- A constructor is used when creating a new object of a particular class
- Constructors are special methods that are called with the **new** operator
 - » **Dog myDog = new Dog("Rover");**
- The name of a constructor is the same as the name of the class
 - » **Dog(String name)** is a constructor for the class **Dog**
- The constructor initializes everything according to its defaults and user supplied parameter values

Local variables

- Local variables are the way that a constructor or method creates little scratchpad notes to use as it performs its responsibility
- Local variables are declared within the body of the constructor or method
 - » “within the body of the method” means “between the curly braces”

```
public void sleep () {  
    double nightLength = 8.5;  
    this area is the body of the method  
}
```
- Local variables can be used exactly as any other variable is used, but they have a limited lifetime

Lifetime of a local variable

- When a constructor or method executes a statement that declares a local variable, a little chunk of memory is made available that it can use for the variable
 - » The variable *is not initialized* until the method does the initialization itself
- The constructor or method can refer to the variable throughout the remainder of the body
- These *local variables* are thrown away when body of the method is finished executing
 - » Next time the method is called, the variables will be allocated fresh
 - » There is no carryover of value from one execution to the next!

Compare Local and Instance Variables

- Local Variable
 - » Defined inside a method
 - » Exists only while the method is being executed
 - » Can be accessed only from the method
 - » Is only meaningful during execution of the method
 - » Contains some intermediate value needed only during execution of the method; its value is not part of the object's state
- Instance Variable
 - » Defined outside any method
 - » Exists as long as the object exists
 - » Can be accessed from any method in the class
 - » Has a meaningful value at any time during the life of the object, whether the object is actively doing something or not
 - » Represents a property of the object; its value is part of the object's state

Document your source code!

- If you write comments as you go along, then the documents are done when the code is done!
 - » This will earn you the eternal gratitude of your boss at work ...
- The **javadoc** tool reads your code and produces fancy html web pages describing it, based on:
 - » information from your comments
 - » information from the structure of the code itself

<http://java.sun.com/j2se/1.4/docs/tooldocs/windows/javadoc.html>

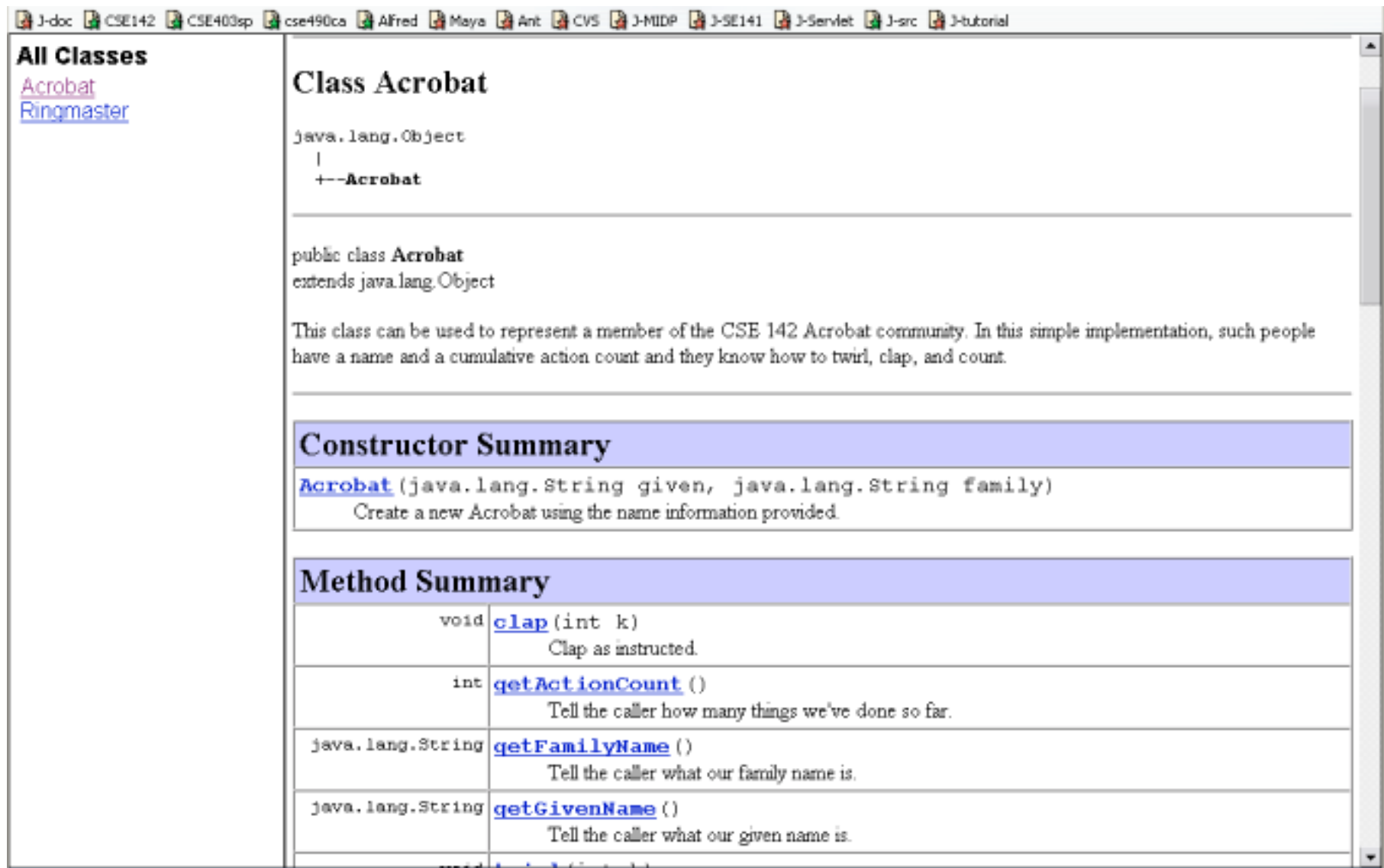
Javadoc Tags

- A javadoc comment applies to the element of the class that follows the comment
- The comment should provide basic information necessary to use the class, the field, or the method
- The javadoc utility supports several “tag” fields in javadoc comments
 - » `@param` -- passed parameter description
 - » `@return` -- returned value description
 - » `@author` -- author
 - » `@throws` -- possible error conditions

These comments ...

```
/**
 * This class can be used to represent a member of the CSE 142 Acrobat
 * community. In this simple implementation, such people have a name and
 * a cumulative action count and they know how to twirl, clap, and count.
 * @author Doug Johnson, for CSE 142 Su03
 */
public class Acrobat {
    /** the specific name of this person */
    private String givenName;
    /** the surname or family name of this person */
    private String familyName;
    /** the cumulative action count for this Acrobat. */
    private int actionCount;
    /**
     * Create a new Acrobat using the name information provided.
     * @param given the specific name of this person
     * @param family the surname or family name of this person
     */
    public Acrobat(String given, String family) {
        givenName = given;
        familyName = family;
        actionCount = 0;
    }
}
```

... produce this documentation



The screenshot shows a web browser window with multiple tabs. The active tab displays the documentation for the `Acrobat` class. On the left, a sidebar titled "All Classes" lists [Acrobat](#) and [Ringmaster](#). The main content area is titled "Class Acrobat" and shows its inheritance from `java.lang.Object`. Below this, the class is defined as `public class Acrobat extends java.lang.Object`. A descriptive paragraph states: "This class can be used to represent a member of the CSE 142 Acrobat community. In this simple implementation, such people have a name and a cumulative action count and they know how to twirl, clap, and count." The documentation is organized into sections: "Constructor Summary" and "Method Summary", both highlighted with blue headers. The constructor summary lists `Acrobat(java.lang.String given, java.lang.String family)` with the description "Create a new Acrobat using the name information provided." The method summary lists four methods: `clap(int k)` (void), `getActionCount()` (int), `getFamilyName()` (java.lang.String), and `getGivenName()` (java.lang.String), each with a brief description of its function.

All Classes
[Acrobat](#)
[Ringmaster](#)

Class Acrobat

`java.lang.Object`
|
+--**Acrobat**

`public class Acrobat`
`extends java.lang.Object`

This class can be used to represent a member of the CSE 142 Acrobat community. In this simple implementation, such people have a name and a cumulative action count and they know how to twirl, clap, and count.

Constructor Summary

[Acrobat](#)(`java.lang.String given, java.lang.String family`)
Create a new Acrobat using the name information provided.

Method Summary

<code>void</code>	clap (<code>int k</code>) Clap as instructed.
<code>int</code>	getActionCount () Tell the caller how many things we've done so far.
<code>java.lang.String</code>	getFamilyName () Tell the caller what our family name is.
<code>java.lang.String</code>	getGivenName () Tell the caller what our given name is.